تعدد المسارات في الفيجول بايزيك دوت نيت

Threading in VB .net

بقلم:

محمد سامر أبو سلو

Samer.selo@yahoo.com

تعدد المسارات

هذا الكتيب يتكلم عن تعدد المسارات في الفيجول بايزيك ويفترض بالقارئ أنه يعرف أساسيات اللغة والفئات والواجهات Classes & Interfaces

ويحتوي على الموضوعات التالية:

- Threading in Windows Forms Applications
- استخدام بحيرة المسارات Using the Thread pool
- تزامن المسارات Thread Synchronization
- كيفية تنفيذ عملية في مسار آخر وإظهار النتيجة في التحكمات على النموذج

Threading in Windows Forms Applications

تكمن المشكلة في أغراض Windows Forms هو أن التحكمات والنموذج ذات نفسه هو أنه يجب الوصول إليهم حصريا من خلال المسار الذي قام بإنشائهم وفي الحقيقة كل أغراض Windows Forms تعتمد على STA Model وذلك بسبب أنها جميعا معتمدة على هيكلية رسائل Win32 والتي ترث مسارات الغرفة Apartment-Threaded مما يعني أنه يمكنك إنشاء النموذج أو التحكم على أي مسار تريده ولكن جميع الطرق المرتبطة به يجب استدعاؤها من نفس المسار. مما يؤدي إلى ظهور العديد من المشاكل بسبب أن أقسام الدوت نيت الأخرى تستخدم Free-Threading model ومزج كلا النوعين بدون حكمة تعتبر فكرة سيئة وحتى لو لم تقم بإنشاء مسار بشكل واضح في كودك ربما ستظهر لك بعض المشاكل في جميع الأحوال فمثلا عندما تحاول الوصول إلى عنصر واجهة مستخدم UI Element من خلال الطريقة Finalize لنوع ما ونحن نعلم أن الطريقة Finalize يتم تنفيذها على مسار مختلف عن المسار الرئيسي

The ISynchronizeInvoke Interface

عناصر التحكم الوحيدة التي يمكنك استدعاؤهم من مسار آخر هم الذين يتم عرضهم من خلال الواجهة ISynchronizeInvoke التي تمتلك الطرائق BeginInvoke و EndInvoke و الخاصية InvokeRequired القابلة للقراءة فقط. حيث تعيد الخاصية InvokeRequired القيمة True إذا كان المستدعي لا يستطيع الوصول إلى التحكم مباشرة وذلك عندما يعمل المستدعي على مسار مختلف عن المسار الذي تم إنشاء التحكم فيه ففي هذه الحالة يجب على المستدعي استدعاء الطريقة Invoke للوصول إلى أي عنصر خاص بالتحكم وهذه الطريقة متزامنة لهذا يتم إيقاف المسار المستدعي حتى يكمل مسار UI تنفيذ الطريقة. أو يمكن للمسار المستدعي استخدام الطرائق BeginInvoke و EndInvoke لتنفيذ العملية بشكل لا متزامن.
تأخذ الطريقة Invoke إجراء مفوض يشير إلى طريقة (Sub أو Function) ويمكنه أخذ مصفوفة من النوع Object كمحدد ثاني إذا كانت الطريقة تتوقع واحد أو أكثر من المحددات وتضمن هيكلية نماذج ويندوز أن الإجراء الذي يشير إليه المفوض يتم تنفيذه في المسار UI لهذا يمكنه بأمان الوصول إلى أي تحكم على النموذج.
سنرى كيف يمكننا استخدام الطريقة Invoke للوصول إلى تحكم من مسار غير المسار UI حيث يظهر لنا المثال التالي كيف يمكننا زيارة جميع المجلدات ضمن شجرة مجلد من مسار ثانوي بينما يتم إظهار اسم المجلد في تحكم Label وأول شئ سنقوم بعمله هو تحديد طريقة تقوم بعمل الإظهار المطلوب التي يمكنها أن تكون مجرد إجراء بسيط

```
' This method must run in the main UI thread.
Sub ShowMessage(ByVal msg As String)
   Me.lblMessage.Text = msg
   Me.Refresh()
End Sub
```

ثم نقوم بتحديد إجراء مفوض يشير لتلك الطريقة ومتغير يحمل كائن لذلك المفوض يكون معرفا على مستوى النموذج كي تتم مشاركته بين جميع الطرائق ضمن النموذج

```
' A delegate that can point to the ShowMessage procedure
Delegate Sub ShowMessageDelegate(ByVal msg As String)
' An instance of the delegate
Dim threadSafeDelegate As ShowMessageDelegate
```

وستحتاج لطريقة تبدأ المسار الثانوي مثلا إجراء معالجة الحدث Click لزر أوامر Button

```
' Parse the c:\Windows directory when the user clicks this button.
Private Sub btnSearch_Click(ByVal sender As Object, ByVal e As EventArgs) _
      Handles btnSearch.Click
   Dim t As New Thread(AddressOf SearchFiles)
   t.Start("c:\windows")
End Sub
```

وأخيرا تقوم بكتابة الكود الذي سيعمل على المسار الثانوي حيث أنه من الضروري لذلك الكود أن يستطيع الوصول للتحكم lblMessage باستدعاء الطريقة ShowMessage وهذا يتم من خلال الطريقة Invoke في فئة النموذج Form Class أو الطريقة Invoke لأي تحكم موجود على النموذج والتي تكون مكافئة لها تماما

```
' (This method runs in a non-UI thread.)
```

```
Sub SearchFiles(ByVal arg As Object)
   ' Retrieve the argument.
   Dim path As String = arg.ToString()
   ' Prepare the delegate
   threadSafeDelegate = New ShowMessageDelegate(AddressOf ShowMessage)
   ' Invoke the worker procedure. (The result isn't used in this demo.)
   Dim files As List(Of String) = GetFiles(path)
   ' Show that execution has terminated.
   Dim msg As String = String.Format("Found {0} files", files.Count)
   Me.Invoke(threadSafeDelegate, msg)
End Sub

' A recursive function that retrieves all the files in a directory tree
' (This method runs in a non-UI thread.)
Function GetFiles(ByVal path As String) As List(Of String)
   ' Display a message.
   Dim msg As String = String.Format("Parsing directory {0}", path)
   Me.Invoke(threadSafeDelegate, msg)
   ' Read the files in this folder and all subfolders.
   Dim files As New List(Of String)
   For Each fi As String In Directory.GetFiles(path)
      files.Add(fi)
   Next
   For Each di As String In Directory.GetDirectories(path)
      files.AddRange(GetFiles(di))
   Next
   Return files
End Function
```

وستتعقد العملية أكثر إن احتجنا لاستخدام الطريقة ShowMessage على جميع المسارات فالطريقة GetFiles مثلا يمكن استدعاؤها من المسار UI وفي هذه الحالة عمل الاستدعاء باستخدام الطريقة Invoke يضيف استباقا للأمور يجب تجنبه لذلك يجب علينا فحص قيمة الخاصية InvokeRequired واستخدام الطريقة العادية إن كانت تعيد القيمة False

```
' (Inside the SearchFiles and GetFiles methods)
If Me.InvokeRequired Then
   Me.Invoke(threadSafeDelegate, msg)
Else
   ShowMessage(msg)
End If
```

والطريقة الأفضل من ذلك بدلا من فحص الخاصية InvokeRequired من أجل كل مستدعي سنقوم بفحصها من داخل الطريقة ShowMessage

```
' This method can run in the UI thread or in a non-UI thread.
Sub ShowMessage(ByVal msg As String)
   ' Use the Invoke method only if necessary.

   If Me.InvokeRequired Then
      Me.Invoke(threadSafeDelegate, msg)
      Return
   End If

   Me.lblMessage.Text = msg
   Me.Refresh()
End Sub
```

فبعد هذا التغيير أي قطعة من الكود ستحتاج لإظهار رسالة على التحكم lblMessage ستحتاج فقط لاستدعاء ShowMessage بدون القلق حول أي مسار يتم تنفيذ الكود عليه

وفي بعض الظروف في فيجول بايزيك 2005 أو الفريموورك رقم 2 يقوم التطبيق بالوصول للتحكم عن طريق مسار غير مسار الإظهار non-UI thread بدون التسبب بأية مشاكل فيمكن حدوث ذلك مثلا عندما تحاول الوصول إلى تحكمات بسيطة مثل Label أو عندما تقوم

بعمليات لا تسبب إرسال رسائل Win32 في الخلفية كما أن العديد من الخصائص يمكن قراءتها وليس تعديلها بدون التسبب بمشاكل وذلك لأن قيمة تلك الخصائص مخزنة في عنصر ضمن تحكم الدوت نيت

The BackgroundWorker Component

على الرغم من أن الواجهة ISynchronizeInvoke تجنبك من الوقوع في المشاكل المتعلقة بالمسارات في تطبيقات نماذج ويندوز يحتاج معظم مطوري فيجول بايزيك لطريقة أفضل وأقل أخطاء فأنت تحتاج مثلا لطريقة بسيطة لإلغاء طريقة غير متزامنة بأسلوب آمن الشئ الذي لا توفره الواجهة المذكورة بشكل تلقائي. ومن أجل هذا السبب قامت مايكروسوفت بإضافة المكون BackgroundWorker إلى صندوق الأدوات واستخدامه سهل جدا مما يسهل عملية إنشاء تطبيقات ويندوز متعددة المسارات.

يمتلك المكون BackgroundWorker خاصيتان مثيرتان للاهتمام فالخاصية WorkerReportsProgress تكون قيمتها True إذا أطلق المكون الحدث ProgressChanged والخاصية WorkerSupportsCancellation تكون قيمتها True إذا كان المكون يدعم الطريقة CancelAsync وتكون القيمة الافتراضية لكلا الطريقتين False لذا يجب عليك ضبط قيمتهم إلى True إذا أردت الاستفادة من جميع مزايا هذا التحكم والمثال الذي سيطرح هنا يفترض أنه قد تم ضبط كلتا القيمتين إلى True ويتطلب استخدام المكون BackgroundWorker بشكل عام العمليات التالية:

١. إنشاء إجراء معالجة للحدث DoWork وملؤها بالكود الذي تريد أن يتم تنفيذه على المسار الثانوي ويتم تشغيل هذا الكود عندما يتم استدعاء الطريقة RunWorkerAsync وهي تقبل محددا يتم تمريره لإجراء معالجة الحدث DoWork حيث لا يمكن للكود الموجود هناك الوصول مباشرة للتحكمات على النموذج لأنه يعمل في مسار آخر
٢. استخدم الطريقة ReportProgress من داخل الحدث DoWork عندما تريد الوصول إلى عنصر على النموذج وهذه الطريقة تطلق الحدث ProgressChanged إذا كانت قيمة الخاصية Worker-ReportsProgress هي True وإلا سيتم إطلاق استثناء Worker-ReportsProgress في حالة كون قيمتها False والكود في إجراء معالجة الحدث ProgressChanged يعمل في نفس المسار UI ولهذا يمكنه الوصول بأمان لأي من تحكمات النموذج
٣. استخدم الطريقة CancelAsync للتحكم BackgroundWorker لإيقاف المسار الثانوي مباشرة وهذه الطريقة تستدعي ضبط الخاصية WorkerSupportsCancellation إلى True وإلا سيتم إطلاق استثناء InvalidOperationException في حالة كون قيمتها False ويجب على الكود في DoWork التحقق دوريا من الخاصية CancellationPending والخروج بأمان عندما تصبح قيمتها True
٤. كتابة إجراء معالجة للحدث RunWorkerCompleted إن كنت تريد القيام بأية أعمال عندما ينتهي عمل المسار الثانوي إما بشكل طبيعي أو بواسطة الإلغاء والكود في إجراء معالجة هذا الحدث يعمل في المسار UI لذا يستطيع الوصول لجميع عناصر النموذج

وبشكل عام فالكود في معالج الحدث DoWork يجب أن يعيد قيمة للمسار الأساسي بدلا من تعيين هذه القيمة في حقل على مستوى الفئة فعلى الكود تعيين هذه القيمة للخاصية Result للغرض DoWorkEventArgs فتكون هذه القيمة متوفرة للمسار الأساسي بواسطة الخاصية Result للغرض RunWorkerCompletedEventArgs الممرر للحدث RunWorkerCompleted وهذا كود نموذجي يستخدم العنصر BackgroundWorker

```
' The button that starts the asynchronous operation
Private Sub btnStart_Click(ByVal sender As Object, ByVal e As EventArgs) _
      Handles btnStart.Click
   Dim argument As Object = "abcde"           ' The argument

   BackgroundWorker1.RunWorkerAsync(argument)
   ' Disable this button, and enable the "Stop" button.
   btnStart.Enabled = False
   btnStop.Enabled = True
End Sub

' The button that cancels the asynchronous operation
Private Sub btnStop_Click(ByVal sender As Object, ByVal e As EventArgs) _
      Handles btnStop.Click
   BackgroundWorker1.RunWorkerAsync(argument)
End Sub
```

```
' The code that performs the asynchronous operation
Private Sub BackgroundWorker1_DoWork(ByVal sender As Object, _
      ByVal e As DoWorkEventArgs) Handles BackgroundWorker1.DoWork
   ' Retrieve the argument.
   Dim argument As Object = e.Argument
   Dim percentage As Integer = 0
   …
   ' The core of the asynchronous task
   Do Until BackgroundWorker1.CancellationPending
      …
      ' Report progress when it makes sense to do so.
      BackgroundWorker1.ReportProgress(percentage)
   Loop
   ' Return the result to the caller.
   e.Result = primes
End Sub

' This method runs when the ReportProgress method is invoked.
Private Sub BackgroundWorker1_ProgressChanged(ByVal sender As Object, _
             ByVal e As ProgressChangedEventArgs) Handles _
             BackgroundWorker1.ProgressChanged

   ' It is safe to access the user interface from here.
   ' For example, show the progress on a progress bar or another control.
   ToolStripProgressBar1.Value = e.ProgressPercentage
End Sub

' This method runs when the asynchronous task is completed (or canceled).
Private Sub BackgroundWorker1_RunWorkerCompleted(ByVal sender As Object, _
                ByVal e As RunWorkerCompletedEventArgs) Handles _
                BackgroundWorker1.RunWorkerCompleted
   ' It is safe to access the user interface from here.
   …
   ' Reset the Enabled state of the Start and Stop buttons.
   btnStart.Enabled = True
   btnStop.Enabled = False
End Sub
```

يظهر لك المثال التالي كيف يمكن استخدام العنصر BackgroundWorker للبحث عن الملفات في مسار غير متزامن وهي نفس المشكلة التي طرحت عند الحديث عن The ISynchronizeInvoke Interface في هذا الموضوع سابقا وبهذا يمكنك مقارنة الطريقتين بسهولة. وستكون النسخة الجديدة المعتمدة على BackgroundWorker أكثر تعقيدا بقليل بسبب أنها تدعم الإلغاء لعمل غير متزامن

```
' The result from the SearchFiles procedure
Dim files As List(Of String)
' We need this variable to avoid nested calls to ProgressChanged.
Dim callInProgress As Boolean

' The same button works as a Start and a Stop button.
Private Sub btnStart_Click(ByVal sender As Object, ByVal e As EventArgs) _
      Handles btnStart.Click
   If btnStart.Text = "Start" Then
      lstFiles.Items.Clear()
      Me.BackgroundWorker1.RunWorkerAsync("c:\windows")
      Me.btnStart.Text = "Stop"
   Else
      Me.BackgroundWorker1.CancelAsync()
   End If
End Sub

' The code that starts the asynchronous file search
Private Sub BackgroundWorker1_DoWork(ByVal sender As Object, _
          ByVal e As DoWorkEventArgs) Handles BackgroundWorker1.DoWork
```

```
    ' Retrieve the argument.
    Dim path As String = e.Argument.ToString()
    ' Invoke the worker procedure.
    files = New List(Of String)
    SearchFiles(path)
    ' Return a result to the RunWorkerCompleted event.
    Dim msg As String = String.Format("Found {0} files", files.Count)
    e.Result = msg
End Sub

' A recursive function that retrieves all the files in a directory tree.
Sub SearchFiles(ByVal path As String)
    ' Display a message.

    Dim msg As String = String.Format("Parsing directory {0}", path)
    ' Notice that we don't really use the percentage;
    ' instead, we pass the message in the UserState property.
    Me.BackgroundWorker1.ReportProgress(0, msg)

    ' Read the files in this folder and all subfolders.
    ' Exit immediately if the task has been canceled.
    For Each fi As String In Directory.GetFiles(path)
       If Me.BackgroundWorker1.CancellationPending Then Return
       files.Add(fi)
    Next
    For Each di As String In Directory.GetDirectories(path)
        If Me.BackgroundWorker1.CancellationPending Then Return
        SearchFiles(di)
    Next
End Sub


Private Sub BackgroundWorker1_ProgressChanged(ByVal sender As Object, _
       ByVal e As ProgressChangedEventArgs) _
       Handles BackgroundWorker1.ProgressChanged
    ' Reject nested calls.
    If callInProgress Then Return
    callInProgress = True
    ' Display the message, received in the UserState property.
    Me.lblMessage.Text = e.UserState.ToString()
    ' Display all files added since last call.
    For i As Integer = lstFiles.Items.Count To files.Count - 1
       lstFiles.Items.Add(files(i))
    Next
    Me.Refresh()
    ' Let the Windows operating system process message in the queue.
    ' If you omit this call, clicks on buttons are ignored.
    Application.DoEvents()
    callInProgress = False
End Sub

Private Sub BackgroundWorker1_RunWorkerCompleted(ByVal sender As Object, _
       ByVal e As RunWorkerCompletedEventArgs) _
       Handles BackgroundWorker1.RunWorkerCompleted
    ' Display the last message and reset button's caption.
    Me.lblMessage.Text = e.Result.ToString()
    btnStart.Text = "Start"
End Sub
```

والكود هنا يشرح نفسه ماعدا إجراء الحدث ProgressChanged حيث يجب أن يتضمن الكود استدعاء للطريقة Application.DoEvents وإلا لن يتمكن التطبيق من معالجة الأحداث المنطلقة مثل حدث النقر على الزر Stop أو أي عمل آخر ممكن إضافته للواجهة ومع ذلك فاستدعاء هذه الطريقة سيسبب استدعاءات معششة للإجراء ProgressChanged مما قد يسبب إطلاق استثناء

StackOverflowException ومن أجل عدم حدوث هذا يتم استخدام حقل منطقي مساعد callInProgress لتجنب حدوث مثل هذه الاستدعاءات المعششة

لاحظ أيضا أن هذا التطبيق لا يحتاج للإعلام عن نسبة التقدم للمسار الرئيسي ويستخدم الطريقة ReportProgress فقط لتنفيذ جزء من الكود في المسار الرئيسي للبرنامج والرسالة الفعلية للإظهار يتم تمريرها للخاصية UserState وإن كان تطبيقك يستخدم progress bar أو أي مؤشر آخر للتقدم يجب عليك تجنب استدعاء الطريقة ReportProgress بدون داعي لأنها تتسبب بتبديل المسارات وتكون مكلفة كثيرا عندما يتعلق الأمر بوقت المعالجة وفي هذه الحالة يجب عليك تخزين مؤشر التقدم في حقل في الفئة واستدعاء الطريقة فقط في حالة حدوث تقدم فعلي

```
Dim currentPercentage As Integer

Private Sub BackgroundWorker1_DoWork(ByVal sender As Object, _
             ByVal e As DoWorkEventArgs) Handles BackgroundWorker1.DoWork

   Const TotalSteps = 5000
   For i As Integer = 1 To TotalSteps
      …
      ' Evaluate progress percentage.
      Dim percentage As Integer = (i * 100) \ TotalSteps
      ' Report to UI thread only if percentage has changed.
      If percentage <> currentPercentage Then
         BackgroundWorker1.ReportProgress(percentage)
         currentPercentage = percentage
      End If
   Next
End Sub
```

Using the Thread pool استخدام بحيرة المسارات

إنشاء العديد من المسارات قد يسبب انخفاض أداء النظام بسرعة وخاصة عندما تصرف هذه المسارات معظم وقتها في حالة سبات أويعاد تشغيلها بصورة دورية بغرض قراءة مصدر ما أو تحديث الإظهار. ولتحسين أداء كودك يمكنك إعادة ترتيب بحيرة المسارات بشكل يضمن أكفأ استخدام للموارد باستخدام بعض الأغراض Objects الموجودة في مجال الأسماء System.Threading بحيرة المسارات

The ThreadPool Type

يتم إنشاء بحيرة المسارات عندما تقوم باستدعاء الدالة ThreadPool.QueueUserWorkItem والتي تحتاج لإجراء مفوض WaitCallback delegate وغرض Object اختياري يستخدم لتمرير البيانات للمسار والإجراء المفوض يجب أن يشير إلى Sub يمرر له محدد وحيد من النوع Object بحيث تكون قيمته محتوية على البيانات التي نريد تمريرها للمسار أو Nothing عندما لا توجد بيانات نريد تمريرها وقطعة الكود التالية تبين لك كيف يمكنك استخدام عدد كبير من المسارات لاستدعاء إجراء في فئة Class

```
For i As Integer = 1 To 20
     ' Create a new object for the next lightweight task.
      Dim task As New LightweightTask()
      ' Pass additional information to it. (Not used in this demo.)
       task.SomeData = "other data"
       ' Run the task with a thread from the pool.
       ' (Pass the counter as an argument.)
       ThreadPool.QueueUserWorkItem(AddressOf task.Execute, i)
 Next
```

وقطعة الكود التالية تحتوي على الكود الذي يتم تنفيذه فعلا عندما يتم سحب المسار من البركة

```
Class LightweightTask
    Public SomeData As String

    ' The method that contains the interesting code
    ' (Not really interesting in this example)
    Sub Execute(ByVal state As Object)
        Console.WriteLine("Message from thread #{0}", state)
    End Sub
End Class
```

والمسار العامل يمكنه تحديد فيما إذا كان قد أخذ من بحيرة المسارات أم لا بتحري قيمة الخاصية Thread.CurrentThread.IsThreadPoolThread ويمكنك معرفة العدد الأقصى للمسارات في البركة باستدعاء الطريقة الساكنة ThreadPool.GetMaxThreads وعدد المسارات المتاحة حاليا باستدعاء الطريقة الساكنة ThreadPool.GetAvailableThreads . كما تم إضافة طريقة جديدة SetMaxThreads في الفريموورك NET Framework 2.0. 2 تمكنك من تغيير العدد الأقصى للمسارات الموجودة في البركة

```
' Maximum 30 worker threads and maximum 10 asynchronous I/O threads in the pool
ThreadPool.SetMaxThreads(30, 10)
```

في بعض الأحيان قد تحتار في نقطة تساؤل هل أقوم بإنشاء المسار بنفسي أم أستعيره من بحيرة المسارات. وتظهر هنا قاعدة جيدة: استخدم فئة المسارات Thread class عندما تريد تنفيذ عملية تريد تنفيذها بأسرع وقت أو عندما تريد القيام بعملية تستهلك الوقت ولا يتم تنفيذها كثيرا وفي معظم الحالات بشكل عام يجب عليك استخدام بحيرة المسارات.

The Timer Type

تقدم الفريموورك عدة أنواع من المؤقتات كل منها يمتلك نقاط قوته وضعفه. فمثلا يجب عليك استخدام التحكم System.Windows.Forms.Timer عندما تقوم بالعمل على تطبيق من النوع Windows Forms applications وإن لم يكن برنامجك يمتلك واجهة للمستخدم يجب عليك عندها استخدام الفئة System.Threading.Timer أو الفئة System.Timers.Timer وتعتبر هاتان الفئتان متساويتين في العمل تقريبا والشرح التالي على System.Threading.Timer ينطبق أيضا على System.Timers.Timer

الفئة Timer في مجال الأسماء System.Threading يقدم طريقة بسيطة لمؤقت يستدعي إجرائية محددة حيث يمكنك استخدام هذه الفئة لجدولة عمل في وقت معين في المستقبل ويمكن تنفيذه بالتكرار الذي تحتاجه مهما يكن ابتداء من مرة واحدة فما فوق وباني المؤقت يأخذ أربعة محددات:

- إجراء مفوض TimerCallback delegate يشير إلى الإجراء الذي يستدعى عندما ينتهي زمن المؤقت ويجب أن يكون هذا الإجراء من النوع Sub يأخذ محدد واحد من النوع Object
- غرض Object يتم تمريره للإجراء الذي يشير إليه المفوض ويمكن أن يكون من عدة أنواع كسلسلة نصية أو مصفوفة أو مجموعة Collection أو أي نوع بيانات آخر يحتوي على البيانات التي سيتم تمريرها للإجراء وإن لم تكن تحتاج لتمرير قيم استخدم Nothing بكل بساطة
- قيمة من النوع TimeSpan تحدد زمن المؤقت الذي سيتم استدعاء الإجراء بعده كما يمكن تحديدها باستخدام قيمة من النوع Long أو UInteger وفي هذه الحالة يقاس الزمن بالميللي ثانية (1/1000 من الثانية) وعند تمرير Timeout.Infinite كقيمة لا يتم إطلاق المؤقت أبدا أو القيمة 0 صفر لإطلاق المؤقت مباشرة
- قيمة من النوع TimeSpan تحدد زمن المؤقت والتي بدورها تحدد زمن تكرار إطلاق المؤقت بعد المرة الأولى. وهذه أيضا يمكن تحديدها بقيمة من النوع Long أو UInteger وهنا أيضا يصبح الوقت مقاسا بالميللي ثانية ويمكنك تمرير القيمة 1 أو Timeout.Infinite لإطلاق المؤقت مرة واحدة فقط.

وهذه القيم التي تمررها لباني المؤقت غير متوفرة كخصائص. وبعد تشغيل المؤقت يمكنك تغيير هذه القيم فقط باستخدام الطريقة Change method والتي تأخذ محددين يحددان وقت التشغيل وفترة زمن المؤقت ويمتلك Timer object إجراء Stop الذي يقوم بإيقاف المؤقت الذي يتم إيقافه عبر استدعاء الإجراء Dispose وترينا قطعة الكود التالية مثالا عما تحدثنا عنه حول المؤقت

```
Sub TestThreadingTimer()
    ' Get the first callback after one second.
    Dim dueTime As New TimeSpan(0, 0, 1)
    ' Get additional callbacks every half second.
    Dim period As New TimeSpan(0, 0, 0, 0, 500)
    ' Create the timer.
    Using t As New Timer(AddressOf TimerProc, Nothing, dueTime, period)
        ' Wait for five seconds in this demo, and then destroy the timer.
        Thread.Sleep(5000)
    End Using
End Sub

' The callback procedure
Sub TimerProc(ByVal state As Object)
    ' Display current system time in console window.
    Console.WriteLine("Callback proc called at {0}", Date.Now)
End Sub
```

وفي النهاية تجدر ملاحظة أن الإجراء المستدعى يتم تنفيذه على مسار مأخوذ من بركة المسارات لذا يجب عليك التحكم بالمتغيرات والمصادر الأخرى المستخدمة من قبل المسار الرئيسي للبرنامج عبر استخدام ما يدعى بتزامن المسارات

تزامن المسارات Thread Synchronization

The SyncLock Statement

خلال زمن التشغيل لا يوجد شئ يضمن لك أن يسير الكود بشكل نظامي بدون مقاطعات وتكون عملية التشغيل بدون مقاطعات عملية قاسية على نظام التشغيل وخاصة عندما يكون عبارة عن بيئة متعددة المهام وفي معظم الحالات التي ستحتاجها ستكون قانعا بالدقة ضمن البرنامج الواحد وذلك عند معالجة الكود فعلى سبيل المثال يكون كافيا لك ضمان أن مسار تنفيذ واحد ضمن التطبيق الحالي يستطيع تنفيذ قطعة معينة من الكود في وقت محدد ويمكنك تحقيق ذلك بتضمين قطعة الكود تلك ضمن كتلة SyncLock…End SyncLock والذي يحتاج إلى متغير كمحدد له محققا المتطلبات التالية:

- يجب أن يكون مشترك بين جميع المسارات ويكون في العادة متغير على مستوى الفئة وبدون الخاصية ThreadStatic
- يجب أن يكون من نوع مرجعي مثل String أو Object واستخدام أنواع القيمة ينتج عنه خطأ في الترجمة
- يجب أن لا يحتوي على القيمة Nothing وفي حال تمرير القيمة Nothing سيسبب أخطاء في زمن التنفيذ

وفيما يلي مثال عن كتلة SyncLock

```
' The lock object. (Any non-Nothing reference value will do.)
Private consoleLock As New Object()

Sub SynchronizationProblem_Task(ByVal obj As Object)
   Dim number As Integer = CInt(obj)
   ' Print a lot of information to the console window.
   For i As Integer = 1 To 1000
      SyncLock consoleLock
         ' Split the output line in two pieces.
         Console.Write(" ")
         Console.Write(number)
      End SyncLock
   Next
End Sub
```

والكود السابق يستخدم المتغير consoleLock للتحكم بالوصول للغرض Console وهو يشكل المصدر الوحيد المشترك بين جميع المسارات في المثال ولهذا فهو المصدر الذي يجب عليك تحقيق التزامن من أجله والتطبيقات الحقيقية يمكن أن تحوي العديد من كتل SyncLock والتي يمكن أن تستخدم نفس المتغير المحلي أو عدة متغيرات مختلفة من أجل اختلاف البصمة وهنا يجب عليك استخدام متغيرا مميزا من أجل كل نوع من أنواع المصادر المشتركة التي يجب عمل التزامن من اجلها أو من أجل مجموعة التعابير التي يجب تنفيذها ضمن المسار في نفس الوقت.

وعندما تستخدم كتلة SyncLock يتضمن الكود تلقائيا كتلة Try…End Try مخفية من أجل ضمان تحرير القفل بشكل صحيح إذا تم إطلاق استثناء ومن أجل هذا لا يمكنك القفز لعبارة داخل الكتلة SyncLock. وإن كانت الكتلة SyncLock موضوعة داخل إجراء خاص بتواجد Instance لفئة ما وجميع المسارات العاملة ضمن إجراء في ذلك التواجد Instance للفئة يمكنك تمرير Me لعبارة الـ SyncLock وذلك بسبب أن هذا الغرض يحقق كل المتطلبات (يمكن الوصول إليه من جميع المسارات – وهو قيمة مرجعية – وبالتأكيد هو ليس Nothing)

```
Class TestClass
   Sub TheTask()
      SyncLock Me
         ' Only one thread at a time can access this code.
         …
      End SyncLock
   End Sub
End Class
```

ملاحظة: يمكنك استخدام Me بهذه الطريقة فقط إن كنت تريد عمل التزامن على مصدر وحيد كملف محدد مثلا أو نافذة الكونسول Console Window وإن كان لديك عدة كتل تزامن التي تحمي عدة مصادر ستستخدم بشكل تلقائي عدة متغيرات كمحددات لكتلة SyncLock. والشئ

الذي له أهمية أكبر مما ذكر هو أنه يجب عليك استخدام Me كمحدد فقط إذا كانت الفئة غير مرئية خارج المجمع الحالي عدا ذلك يمكن لتطبيق آخر استخدام نفس التواجد Instance للفئة ضمن كتلة SyncLock مختلفة وبهذا فلن يتم تنفيذ عدة كتل من الكود بدون سبب حقيقي محدد وبشكل عام لا يجب عليك استخدام غرض Object عام مرئي من مجمعات أخرى كمحدد لكتلة SyncLock. وتجدر الملاحظة أن العديد من الأكواد التي تراها على الانترنت تستخدم العامل GetType للحصول على نوع الغرض المستخدم للقفل lock object وذلك لحماية الطريقة الساكنة.

عندما تستخدم عبارات SyncLock معششة للقيام بالتزامن لأغراض مختلفة من الضروري استخدام تسلسل تعشيش متطابق أينما احتجت له في تطبيقك فالتحري عن الأقفال بالتسلسل المطابق ذاته يجنبك الوصول إلى حالة الأقفال الميتة خلال العديد من أجزاء التطبيق وهذه القاعدة تنطبق أيضا عندما تقوم دالة تحتوي على SyncLock باستدعاء دالة أخرى تحتوي على SyncLock

```
' Always use this sequence when locking objLock1 and objLock2.
SyncLock objLock1
   SyncLock objLock2
      …
   End SyncLock
End SyncLock
```

اعتبارات الأداء والتواجد الكسول Performance Considerations and Lazy Instantiation

تضمين جميع الأكواد التي تستخدم متغيرات مشتركة ضمن كتلة SyncLock يؤدي إلى إبطاء تطبيقك كثيرا أو تخفيض أداؤه بشكل ملحوظ وبشكل خاص عندما يتم تشغيله على حاسب متعدد المعالجات فإن استطعت تجنب استخدام كتلة SyncLock بدون تعريض تكامل البيانات للخطر يجب عليك القيام به قطعيا فمثلا تخيل أنك تستخدم نمط وحيد بتواجد كسول lazy instantiation في بيئة متعددة المسارات

```
Public Class Singleton
   Private Shared m_Instance As Singleton
   Private Shared sharedLock As New Object()

   Public Shared ReadOnly Property Instance() As Singleton
      Get
         SyncLock sharedLock
            If m_Instance Is Nothing Then m_Instance = New Singleton
            Return m_Instance
         End SyncLock
      End Get
   End Property
End Class
```

تكمن المشكلة في الكود السابق أن معظم الوصولات للخاصية لا يحتاج إلى تزامن وذلك لأن المتغير الخاص m_Instance يتم تعيينه مرة واحدة في المرة الأولى التي يتم فيها قراءة الخاصية وفي ما يلي طريقة أفضل لتحقيق التصرف المطلوب

```
Class Singleton
   Private Shared m_Instance As Singleton
   Private Shared sharedLock As New Object

   Public Shared ReadOnly Property Instance() As Singleton
      Get
         If m_Instance Is Nothing Then
            SyncLock sharedLock
               If m_Instance Is Nothing Then m_Instance = New Singleton()
            End SyncLock
         End If
         Return m_Instance
      End Get
   End Property
End Class
```

الأغراض المتزامنة Synchronized Objects

مشكلة أخرى متعلقة بالمسارات في الدوت نيت هي أن ليس جميع أغراض الدوت نيت NET object. قابلة للمشاركة بأمان عبر المسارات not all .NET objects are thread-safe فعندما تقوم بكتابة تطبيق متعدد المسارات يجب عليك التأكد دوما من الوثائق للتأكد من أن الأغراض والطرائق التي تستخدمها آمنة للاستخدام عبر المسارات فعلى سبيل المثال جميع الطرق الساكنة للفئات Regex و Match أمنة عبر المسارات ولكن الطرق الغير ساكنة غير آمنة فيجب عدم استخدامها ضمن مسار مختلف وكذلك بعض أغراض الدوت نيت مثل Windows Forms objects and controls لها العديد من الحدود التي تجعل فقط المسار الذي أنشأها يمكنه استدعاء طرقها وخصائصها

أنواع دوت نيت المتزامنة Synchronized .NET Types

العديد من الأغراض الغير آمنة عبر المسارات بطبيعتها مثل ArrayList و Hashtable و Queue و SortedList و Stack و TextReaderو TextWriter و التعابير النظامية تقدم طريقة ساكنة قابلة للتزامن تعيد غرض أمن للمسارات thread-safe object مكافئ للذي تم تمريره كما أن معظمها يعرض الخاصية IsSynchronized التي تعيد True عندما تتعامل مع نسخة آمنة عبر المسارات

```
' Create an ArrayList object, and add some values to it.
Dim al As New ArrayList()
al.Add(1): al.Add(2): al.Add(3)
' Create a synchronized, thread-safe version of this ArrayList.
Dim syncAl As ArrayList = ArrayList.Synchronized(al)
' Prove that the new object is thread-safe.
Console.WriteLine(al.IsSynchronized)         ' => False
Console.WriteLine(syncAl.IsSynchronized)     ' => True
' You can now share the syncAl object among different threads
```

تذكر دائما أن التعامل مع هذه النسخة المتزامنة يكون أبطأ من النسخة الغير متزامنة وذلك بسبب أن كل طريقة تمر عبر سلسلة من الفحوصات الداخلية وفي معظم الحالات يمكنك كتابة كود فعال أكثر إذا استخدمت المصفوفات والمجموعات العادية regular arrays and collections وقمت بمزامنة عناصرها باستخدام كتلة SyncLock العادية

The Synchronization Attribute

استخدام الخاصية System.Runtime.Remoting.Contexts.Synchronization هي أبسط طريقة لتحقيق الوصول المتزامن للغرض Object بأكمله وبذلك يستطيع مسار واحد فقط الوصول إلى حقوله وطرائقه وبذلك أي مسار يستطيع استخدام الفئة ولكن مسار واحد فقط يستطيع تنفيذ أحد طرائقه إذا كانت الطريقة تنفذ كودا ضمن الفئة Class وأي مسار يحاول استخدام هذه الفئة عليه الانتظار وبكلمات أخرى وكأن هناك كتل SyncLock تغلف كافة طرائق الفئة مستخدمة نفس متغير الإقفال. والكود التالي يبين كيف يمكنك مزامنة فئة باستخدام الخاصية Synchronization attribute لاحظ أيضا أن الفئة يجب أن يتم وراثتها من ContextBoundObject ليتم تعليمها كـ context-bound object

```
System.Runtime.Remoting.Contexts.Synchronization()> _
Class Display
   Inherits ContextBoundObject
   …
End Class
```

و خاصية التزامن Synchronization attribute تضمن الوصول المتزامن لجميع الحقول والخصائص والطرق ولكنها لا توفر التزامن للأعضاء الساكنين static members وهي تأخذ محددا اختياريا يمكن أن تكون قيمته True أو False أو أحد الثوابت التي توفرها الفئة SynchronizationAttribute والتي يمكنك الاطلاع عليها من مكتبة MSDN

The MethodImpl Attribute

في معظم الحالات مزامنة فئة كاملة ستقتل التطبيق وحماية بعض الطرائق في تلك الفئة يكون كافيا في معظم الحالات حيث يمكنك تطبيق هذا بتغليف كود الطريقة بكتلة SyncLock أو يمكنك استخدام تقنية أبسط مبنية على الصفة System.Runtime.CompilerServices.MethodImpl

```
Class MethodImplDemoClass
    ' This method can be executed by one thread at a time.
    <MethodImpl(MethodImplOptions.Synchronized)> _
    Sub SynchronizedMethod()
      …
    End Sub
End Class
```

فتطبيق الصفة MethodImpl على عدة طرائق في الفئة يؤدي نفس الغرض من تغليف كامل تلك الطرائق بكتلة SyncLock والتي تستخدم Me كمتغير إقفال وبكلمات أخرى أي مسار يستدعي طريقة معلمة بالخاصية MethodImpl سوف يمنع أي مسار آخر من استدعاء الطريقة المعلمة بالخاصية MethodImpl كما يمكنك استخدام هذه الخاصية على الطرائق الساكنة ويكون متغير الغرض الذي يستخدم ضمنيا لقفل الطرائق الساكنة مختلف عن متغير الغرض المستخدم لقفل الطرائق الأخرى للفئة instance methods وبهذا فالمسار الذي يستدعي طريقة ساكنة معلمة بالصفة MethodImpl لا يمنع مسار آخر من استدعاء الطرائق الغير ساكنة instance methods والمعلمة بنفس الصفة

عمليات القراءة والكتابة المتغيرة Volatile Read and Write Operations

عندما تتم مشاركة متغير عبر عدة مسارات والتطبيق يعمل على حاسب متعدد المعالجات يجب عليك وضع احتمال حدوث أخطاء إضافية في الحسبان وتكمن المشكلة في النظام متعدد المعالجات في أن لكل معالج الكاش الخاص به ولهذا فإذا قمت بالكتابة على حقل في فئة على مسار سيتم كتابة القيمة الجديدة في الكاش المرتبط مع المعالج الحالي ولا يتم نشرها مباشرة إلى الكاش الخاص ببقية المعالجات بحيث يمكنهم جميعا رؤية القيمة الجديدة. كما تحدث مشكلة مشابهة في الأنظمة ذات المعالج 64 بت الذي يمكنه إعادة ترتيب تنفيذ كتل عبارات الكود متضمنا عمليات القراءة والكتابة في الذاكرة وعملية إعادة الترتيب لم يكن لها تأثير ظاهر حتى الآن من أجل مسار واحد يستخدم جزءا معينا من الذاكرة ولكن ربما سيسبب ذلك مشكلة عندما يتم الوصول إلى نفس الجزء من الذاكرة بواسطة عدة مسارات. وتوفر الفريموورك حلان لهذه المشكلة وهما زوج من الطرائق VolatileRead و VolatileWrite والطريقة MemoryBarrier ويوفرها جميعا النوع Thread

تمكنك الطريقة VolatileWrite من كتابة متغير والتأكد من أن القيمة الجديدة يتم كتابتها آليا في الذاكرة المشتركة بين جميع المعالجات ولا تبقى في المسجل الخاص بالمعالج حيث تكون مخفية عن بقية المسارات وبالمثل تمكنك الطريقة VolatileRead من قراءة المتغير بطريقة آمنة لأنها تجبر النظام على تفريغ جميع ذواكر الكاش الموجودة قبل تنفيذ العملية وكلا الطريقتان محملتان تحميلا زائدا Overloaded بحيث تأخذ متغيرات رقمية أو غرضية Object وبالمرجع كما في قطعة الكود التالية

```
Class TestClass
    Private Shared sharedValue As Integer

    Function IncrementValue() As Integer
        Dim value As Integer = Thread.VolatileRead(sharedValue)
        value += 1
        Thread.VolatileWrite(sharedValue, value)
        Return value
    End Function
End Class
```

والطريقتان المذكورتان تعملان بشكل جيد عندما نتعامل مع المتغيرات الرقمية أو الغرضية Object ولكن لا يمكن استخدامهما من أجل أنواع أخرى من المتغيرات لأنه لا يمكنك استخدام نسخة الدالة التي تأخذ متغير من النوع Object بسبب عدم إمكانية الاعتماد على عملية التحويل عندما يكون المتغير ممررا بالمرجع مما يقودنا إلى الطريقة MemoryBarrier التي تقوم بتفريغ محتويات جميع ذواكر الكاش

الخاصة بالمعالجات إلى الذاكرة الرئيسية وبهذا تضمن لك أن جميع المتغيرات تحتوي أحدث نسخة من البيانات التي تمت كتابتها إليهم فمثلا يضمن الكود التالي أن الفئة Singleton تعمل جيدا حتى على نظام متعدد المعالجات

```
Class Singleton
    Private Shared m_Instance As Singleton
    Private Shared sharedLock As New Object()

    Public Shared ReadOnly Property Instance() As Singleton
        Get
            If m_Instance Is Nothing Then
                SyncLock sharedLock
                    If m_Instance Is Nothing Then
                        Dim tempInstance As Singleton = New Singleton()
                       ' Ensure that writes related to instantiation are flushed.
                        Thread.MemoryBarrier()
                        m_Instance = tempInstance
                    End If
                End SyncLock
            End If
            Return m_Instance
        End Get
    End Property
End Class
```

ويجب عليك استدعاء الطريقة MemoryBarrier مباشرة قبل أن يتم نشر القيمة الجديدة إلى بقية المسارات وفي المثال السابق يتم التأكد من اكتمال وضع القيمة في المتغير tempInstance قبل أن توضع في المتغير الذي ستتم مشاركته عبر المسارات

The Monitor Type

توفر كتلة SyncLock طريقة سهلة لاستخدام طريقة تتعامل مع مسائل التزامن ولكنها تكون غير ملائمة في العديد من الحالات فمثلا لا يمكن للمسار اختبار كود في كتلة SyncLock وتجنب منعه من ذلك إذا كان مسار آخر ينفذ كتلة SyncLock مرتبطة مع نفس الغرض Object وكتل SyncLock معرفة داخليا بواسطة Monitor objects التي يمكن استخدامها مباشرة للحصول على مرونة أكثر ويتم ذلك على حساب زيادة التعقيد في الكود. ولا يمكنك استخدام Monitor object وحيد وفي الحقيقة جميع طرق Monitor type التي سيتم عرضها هي طرائق ساكنة وتعتبر Enter هي الطريقة الأهم وهي تأخذ محدد من النوع Object الذي يعمل كالمحدد الممرر لكتلة SyncLock وتكون له نفس الشروط من كونه من نوع مرجعي ومشترك ولا يمكن أن يحمل القيمة Nothing وإن لم تمتلك المسارات الأخرى قفلا على هذا الغرض فيقوم المسار الحالي بطلب ذلك القفل ويضبط قيمة العداد إلى 1 وإن امتلك مسار آخر القفل يجب على المسار الطالب انتظار أن يقوم المسار الآخر بتحرير القفل حتى يصبح متوفرا وإن كان المسار الطالب يمتلك القفل أساسا يؤدي كل استدعاء للطريقة Monitor.Enter إلى زيادة قيمة العداد. وتأخذ الطريقة Monitor.Exit غرض القفل lock object كمحدد لها وتنقص قيمة العداد وعندما تصل قيمة العداد للصفر يتم تحرير القفل ممكنا بقية المسارات من الحصول عليه ويجب أن يتم الموازنة بين استدعاء الطريقة Monitor.Enter والطريقة Monitor.Exit أو لن يتم تحرير القفل أبدا

```
' A non-Nothing module-level object variable
Dim objLock As New Object()
…
Try
   ' Attempt to enter the protected section;
   ' wait if the lock is currently owned by another thread.
   Monitor.Enter(objLock)
   ' Do something here.
   …
Finally
   ' Release the lock.
   Monitor.Exit(objLock)
End Try
```

إذا كان هناك احتمال في أن تطلق العبارات الموجودة بين الطريقتان Enter و Exit استثناء يجب عليك عندها وضع كامل الكود ضمن كتلة Try...End Try لأنه من الضروري أن تقوم بتحرير القفل دوما وإن طلب مسار طريقة على مسار آخر تنتظر داخل الطريقة Monitor.Enter سوف يستقبل ذلك المسار استثناء ThreadInterruptedException الذي يعتبر سببا إضافيا لاستخدام كتلة Try...End Try والطريقتان Enter و Exit الخاصتين بـ Monitor Object يسمحان لك باستبدال كتلة SyncLock ولكنهما لا يقدمان لك أية فوائد إضافية وسوف ترى المرونة الزائدة للفئة Monitor عندما تطبق الطريقة TryEnter وهي مشابهة للطريقة Enter ولكنها تخرج وتعيد False إذا كان لا يمكن الحصول على القفل خلال فترة زمنية محددة فمثلا يمكنك محاولة الحصول على Monitor خلال 10 ميللي ثانية ثم التخلي عن ذلك دون أن توقف المسار الحالي مدة غير محددة ويقوم الكود التالي بإعادة كتابة المثال السابق المعتمد على SyncLock مستخدما Monitor object ويظهر لك المحاولات الفاشلة للحصول على القفل

```
Try
   Do Until Monitor.TryEnter(consoleLock, 10)
      Debug.WriteLine("Thread " + Thread.CurrentThread.Name + _
         " failed to acquire the lock")
   Loop
   ' Split the output line in pieces.
   Console.Write(" ")
   Console.Write(Thread.CurrentThread.Name)
Finally
   ' Release the lock.
   Monitor.Exit(consoleLock)
End Try
```

The Mutex Type

يوفر النوع Mutex مبدأ آخر للتزامن حيث أن الـ Mutex هو Windows kernel object يمكن امتلاكه من قبل مسار واحد فقط في الوقت نفسه ويكون في حالة إشارة a signaled state عندما لا يمتلكه أي مسار. ويطلب المسار ملكية الـ Mutex باستخدام الطريقة الساكنة Mutex.WaitOne والتي لا تعود إلا بعد أن يتم تحقيق الملكية ويتم تحريرها باستخدام الطريقة الساكنة Mutex.ReleaseMutex والمسار الذي يطلب ملكية Mutex object المملوك من قبله سلفا لا يمنع نفسه من الحصول على الملكية فيجب عليك في هذه الحالة استدعاء ReleaseMutex بعدد مساوي من المرات وهذا مثال عن كيفية تعريف قسم متزامن باستخدام Mutex object

```
' This Mutex object must be accessible to all threads.
Dim m As New Mutex()

Sub WaitOneExample()
   m.WaitOne()
   ' Enter the synchronized section.
   …
   ' Exit the synchronized section.
   m.ReleaseMutex()
End Sub
```

وفي التطبيقات الحقيقية عليك استخدام كتلة Try لحماية كودك من الأخطاء ووضع استدعاء ReleaseMutex في قسم Finally وإن قمت بتمرير محدد اختياري للطريقة WaitOne كزمن انتهاء فستعيد التحكم للمسار عندما يتم تحقيق الملكية بنجاح أو عندما ينتهي الوقت المحدد ويمكن معرفة الفرق بين النتيجتين باختبار القيمة المعادة حيث أن True تعني تحقيق الملكية و False تعني انتهاء الوقت

```
' Attempt to enter the synchronized section, but give up after 0.1 seconds.
If m.WaitOne(100, False) Then

   ' Enter the synchronized section.
   …
   ' Exit the synchronized section, and release the mutex.
   m.ReleaseMutex()
```

```
End If
```

عند استخدام هذه الطريقة يوفر النوع Mutex آلية مكافئة للطريقة Monitor.TryEnter بدون تقديم أية خصائص إضافية ويمكنك رؤية المرونة الإضافية للنوع Mutex عندما ترى الطريقتين الساكنتين WaitAny و WaitAll الخاصتين به والطريقة WaitAny تأخذ مصفوفة من Mutex objects وتعود عندما تحقق ملكية واحد من Mutex objects من تلك القائمة وفي هذه الحالة يصبح الـ Mutex في حالة إشارة أو عندما ينتهي الوقت المحدد بالمحدد الاختياري والقيمة المعادة تكون عبارة عن مصفوفة من Mutex objects التي أصبحت في حالة إشارة أو قيمة خاصة هي 258 عندما ينتهي الوقت المحدد. وتستخدم مصفوفة من Mutex objects عندما يكون لدينا عدد محدود من الموارد ونريد أن نربط كل واحد منها بمسار حالما يصبح ذلك المصدر متوفرا وفي هذه الحالة يصبح الـ Mutex objects الذي في حالة إشارة يعني أن المصدر الموافق متوفر عندئذ يمكنك استخدام الطريقة Mutex.WaitAny لمنع المسار الحالي حتى يصبح واحدا من الـ Mutex objects في حالة إشارة و النوع Mutex يرث الطريقة WaitAny من WaitHandle الخاصة بفئته الأب وهذا هيكل لتطبيق يستخدم هذه التقنية

```
' An array of three Mutex objects
Dim mutexes() As Mutex = {New Mutex(), New Mutex(), New Mutex()}

Sub WaitAnyExample()
   ' Wait until a resource becomes available.
   ' (Returns the index of the available resource.)
   Dim mutexNdx As Integer = Mutex.WaitAny(mutexes)
   ' Enter the synchronized section.
   ' (This code should use only the resource corresponding to mutexNdx.)
   …
   ' Exit the synchronized section, and release the resource.
   mutexes(mutexNdx).ReleaseMutex()
End Sub
```

والطريقة الساكنة WaitAll أيضا موروثة من WaitHandle الخاصة بالفئة الأب حيث تأخذ مصفوفة من Mutex objects وتعيد التحكم للتطبيق فقط عندما يصبح جميعهم في حالة إشارة وهي مفيدة بشكل خاص عندما لا يمكنك المتابعة إلا عندما تكون جميع المسارات الباقية قد أنهت عملها

```
' Wait until all resources have been released.
Mutex.WaitAll(mutexes)
```

وهناك مشكلة صغيرة متعلقة بالطريقة WaitAll هي أنه لا يمكن استدعاؤها من المسار الرئيسي في تطبيق مسار الغرفة الوحيدة Single Thread Apartment (STA) application مثل تطبيق الكونسول Console application أو تطبيق نماذج ويندوز Windows Forms application ففي المسار الرئيسي لتطبيق STA يجب عليك التوقف حتى يتم تحرير مجموعة من الـ Mutex عندها يجب عليك استخدام WaitAll من مسار منفصل ثم استخدام الطريقة Thread.Join على ذلك المسار لإيقاف المسار الرئيسي حتى تعود الطريقة WaitAll وفي فيجول بايزيك 2005 والنسخة 2 من الفريموورك يوجد الطريقة الساكنة الجديدة SignalAndWait تمكنك من وضع Mutex object في حالة إشارة وانتظار Mutex object آخر

```
' Signal the first mutex and wait for the second mutex to become signaled.
Mutex.SignalAndWait(mutexes(0), mutexes(1))
```

وخلافا لجميع أغراض التزامن التي تم ذكرها حتى الآن يمكن لـ Mutex objects أن يرتبط باسم الأمر الذي يعد من أهم المزايا لهذه الأغراض فأغراض Mutex objects التي تمتلك نفس الاسم يمكن مشاركتها عبر العمليات ويمكنك إنشاء تواجد Instance لها كما يلي

```
Dim m As New Mutex(False, "mutexname")
```

وإن كان الاسم موجودا سابقا في النظام يحصل المستدعي على مرجع له وإلا سيتم إنشاء Mutex object جديد بحيث تمكنك هذه الآلية من مشاركة Mutex objects عبر عدة تطبيقات مختلفة وبهذا تتمكن هذه التطبيقات من مزامنة عمليات الوصول للمصادر المختلفة وقد تم إضافة باني جديد في الفريموورك 2 وفيجول بايزيك 2005 يمكنك من اختبار إذا كان قد تم منح المسار المستدعي ملكية الـ Mutex

```
Dim ownership As Boolean
Dim m As New Mutex(True, "mutexname", ownership)
If ownership Then
   ' This thread owns the mutex.
   …
End If
```

من الاستخدامات الشائعة لـ named mutexes هو تحديد فيما إذا كان التطبيق العامل هو الأول أو الوحيد الذي تم تحميله وإن لم تكن هذه الحالة يمكن للتطبيق الخروج مباشرة أو الانتظار حتى تنتهي النسخة الأخرى من مهامها كما في المثال

```
Sub Main()
   Dim ownership As Boolean
   Dim m As New Mutex(True, "DemoMutex", ownership)
   If ownership Then
      Console.WriteLine("This app got the ownership of Mutex named DemoMutex")
      Console.WriteLine("Press ENTER to run another instance of this app")
      Console.ReadLine()
      Process.Start(Assembly.GetExecutingAssembly().GetName().CodeBase)
   Else
      Console.WriteLine("This app is waiting to get ownership of Mutex named 
DemoMutex")
      m.WaitOne()
   End If
   ' Perform the task here.
   …
   Console.WriteLine("Press ENTER to release ownership of the mutex")
   Console.ReadLine()
   m.ReleaseMutex()
End Sub
```

والطريقة الساكنة OpenExisting جديدة أيضا في الفريموورك 2 وتقدم طريقة أخرى لفتح Mutex على مستوى النظام named system-wide Mutex object وبعكس باني الـ Mutex تمكنك هذه الطريقة من تحديد درجة التحكم التي تريدها على الـ Mutex

```
Try
   ' Request a mutex with the right to wait for it and to release it.
   Dim rights As MutexRights = MutexRights.Synchronize Or MutexRights.Modify
   Dim m As Mutex = Mutex.OpenExisting("mutexname", rights)
   ' Use the mutex here.
   …
Catch ex As WaitHandleCannotBeOpenedException
   ' The specified object doesn't exist.
Catch ex As UnauthorizedAccessException
   ' The specified object exists, but current user doesn't have the
   ' necessary access rights.
Catch ex As IOException
   ' A Win32 error has occurred.
End Try
```

وفي فيجول بايزيك 2005 والفريموورك 2 تظهر الميزة الجديدة الأهم في النوع Mutex وهي إمكانية الوصول لقوائم التحكم بالوصول access control lists (ACLs) في النموذج عبر الغرض System.Security.AccessControl.MutexSecurity object حيث

يمكنك تحديد ACL عندما تنشئ غرض Mutex جديد مستخدما الطريقة GetAccessControl للحصول على غرض MutexSecurity المرتبط بـ Mutex محدد وتطبيق ACL جديد باستخدام الطريقة SetAccessControl

```
Dim ownership As Boolean
Dim m As New Mutex(True, "mutexname", ownership)
If Not ownership Then
   ' Determine who is the owner of the mutex.
   Dim mutexSec As MutexSecurity = m.GetAccessControl()
   Dim account As NTAccount = DirectCast(mutexSec.GetOwner( _
   GetType(NTAccount)), NTAccount)
   Console.WriteLine("Mutex is owned by {0}", account)
End If
```

The Semaphore Type

تقدم الفريموورك 2 و فيجول بايزيك دوت نيت نوعا جديدا وهو Semaphore type الذي يرتكز على Win32 semaphore object وخلافا لبقية أغراض المسارات الموجودة في المكتبة mscorlib فهذا النوع تم تعريفه في المكتبة system.dll وهو يستخدم عندما تريد تحديد حد أقصى (عدد N) من المسارات التي يمكن تنفيذها في جزء معين من الكود أو للوصول إلى مصدر معين ويمتلك عددا ابتدائيا وعددا أقصى ويجب عليك تمرير هذه القيم لبانيه

```
' A semaphore that has an initial count of 1 and a maximum count of 2.
Dim sem As New Semaphore(1, 2)
```

يحاول المسار أخذ ملكية الـ semaphore باستدعاء الطريقة WaitOne وإن كان العدد الحالي أكبر من الصفر يتم إنقاصه وتعود الطريقة مباشرة وإلا تنتظر حتى يحرر مسار آخر semaphore أو إنقضاء الوقت المحدد بالمحدد الاختياري ويحرر المسار semaphore باستدعاء الطريقة Release مما يزيد العدد بمقدار 1 أو بقيمة محددة ويعيد قيمة العدد السابق

```
Dim sem As New Semaphore(2, 2)
' Next statement brings count from 2 to 1.
sem.WaitOne()
…
' Next statement brings count from 1 to 2.
sem.Release()
' Next statement attempts to bring count from 2 to 3, but
' throws a SemaphoreFullException.
sem.Release()
```

وبشكل أساسي ستستخدم الغرض Semaphore كما يلي

```
' Initial count is initially equal to max count.
Dim sem2 As New Semaphore(2, 2)

Sub Semaphore_Example()
   ' Wait until a resource becomes available.
   sem2.WaitOne()
   ' Enter the synchronized section.
   …
   ' Exit the synchronized section, and release the resource.
   sem2.Release()
End Sub
```

تذكر دوما استخدام الكتلة Try…Finally للتأكد من أن الـ semaphore قد تم تحريره حتى لو حدث استثناء ما وتماما كالـ mutexes يمكن للـ semaphores امتلاك اسم ومشاركته عبر العمليات وعندما تحاول إنشاء غرض semaphores موجود سابقا يتم تجاهل العدد والحد الأقصى

```
Dim ownership As Boolean
Dim sem3 As New Semaphore(2, 2, "semaphoreName", ownership)
If ownership Then
   ' Current thread has the ownership of the semaphore.
   …
End If
```

ويدعم الغرض Semaphore أيضا الـ ACLs التي يمكن تمريرها للباني حيث تتم القراءة بواسطة الطريقة GetAccessControl والتعديل بواسطة الطريقة SetAccessControl ومن الضروري أن تلاحظ أن النوع Mutex والنوع Semaphore تتم وراثتهما من الفئة الأساسية WaitHandle لذا يمكن تمريرهما كمحددات للطرائق الساكنة WaitAny و WaitAll و SignalAndWait للنوع WaitHandle مما يمكنك من مزامنة المصادر بسهولة والتي تكون محمية بواسطة أيا من هذه الأغراض كما في الكود

```
' Wait until two mutexes, two semaphores, and one event object become signaled.
Dim waitHandles() As WaitHandle = {mutex1, mutex2, sem1, sem2, event1}
WaitHandle.WaitAll(waitHandles)
```

The ReaderWriterLock Type

العديد من المصادر في العالم الحقيقي يمكن إما القراءة منها أو الكتابة إليها وهي تدعم في الغالب إما مجموعة قراءات متعددة أو عملية كتابة وحيدة يتم تنفيذها في لحظة معينة فمثلا يمكن لعدة عملاء القراءة من ملف بيانات أو جدول في قاعدة بيانات ولكن إن تمت الكتابة للملف أو الجدول فلا يمكن حدوث أي عمليات قراءة أو كتابة على ذلك المصدر حيث يمكنك تعريف قفلا لكتابة واحدة أو عدة قراءات بدلالة الغرض ReaderWriterLock واستخدام هذا الغرض يعتبر رؤية إلى الأمام فكل المسارات التي تريد استخدام مصدر معين يجب عليها استخدام نفس الغرض ReaderWriterLock وقبل محاولة القيام بأي عملية على ذلك المصدر يجب على المسار استدعاء إما الطريقة AcquireReaderLock أو الطريقة AcquireWriterLock وذلك اعتمادا على العملية التي يتم تنفيذها وهذه الطرائق تقوم بإيقاف المسار الحالي حتى يتم الحصول على ذلك المصدر وأخيرا على المسار استدعاء الطريقة ReleaseReaderLock أو الطريقة ReleaseWriterLock عندما تنهي عملية القراءة أو الكتابة على ذلك المصدر والمثال التالي يقوم بإنشاء 10 مسارات تقوم بعملية قراءة أو كتابة على مصدر مشترك

```
Dim rwl As New ReaderWriterLock()
Dim rnd As New Random()

Sub TestReaderWriterLock()
   For i As Integer = 0 To 9
      Dim t As New Thread(AddressOf ReaderWriterLock_Task)
      t.Start(i)
   Next
   …
End Sub

Sub ReaderWriterLock_Task(ByVal obj As Object)
   Dim n As Integer = CInt(obj)
   ' Perform 10 read or write operations. (Reads are more frequent.)
   For i As Integer = 1 To 10
      If rnd.NextDouble < 0.8 Then
         ' Attempt a read operation.
         rwl.AcquireReaderLock(Timeout.Infinite)
         Console.WriteLine("Thread #{0} is reading", n)
         Thread.Sleep(300)
         Console.WriteLine("Thread #{0} completed the read operation", n)
         rwl.ReleaseReaderLock()
      Else
```

```
        ' Attempt a write operation.
        rwl.AcquireWriterLock(Timeout.Infinite)
        Console.WriteLine("Thread #{0} is writing", n)
        Thread.Sleep(300)
        Console.WriteLine("Thread #{0} completed the write operation", n)
        rwl.ReleaseWriterLock()
     End If
  Next
End Sub
```

وعندما تشغل هذا الكود سترى أن عدة مسارات يمكنها القراءة بنفس الوقت والمسار الذي يقوم بالكتابة يوقف جميع المسارات الأخرى ويمكن للطريقتان AcquireReaderLock و AcquireWriterLock أخذ محدد عبارة عن زمن انتهاء timeout وذلك بقيمة من النوع TimeSpan أو بعدد من الميللي ثانية ويمكنك اختبار فيما إذا تم الحصول على القفل بنجاح باستخدام الخصائص IsReaderLockHeld و IsWriterLockHeld القابلة للقراءة فقط إذا مررت قيمة غير Timeout.Infinite

```
' Attempt to acquire a reader lock for no longer than 1 second.
rwl.AcquireWriterLock(1000)
If rwl.IsWriterLockHeld Then
   ' The thread has a writer lock on the resource.
   …
End If
```

والمسار الذي يمتلك قفل القراءة يمكنه الترقية إلى قفل للكتابة باستدعاء الطريقة UpgradeToWriterLock والعودة ثانية لوضع القراءة باستخدام الطريقة Downgrade-FromWriterLock والشئ الرائع بخصوص الأغراض ReaderWriterLock هي أنها أغراض خفيفة بحيث يمكن استخدامها عددا كبيرا من المرات دون أن تؤثر على الأداء بشكل ملحوظ وبما أن الطرائق AcquireReaderLock و AcquireWriterLock تأخذان وقت انتهاء فالتطبيق المصمم بشكل جيد يجب أن لا يعاني من أقفال ميتة ومع ذلك يمكن حصول حالة قفل ميت عندما يكون مساران ينتظران مصدرا محجوزا من قبل مسار لا يقوم بتحريره حتى انتهاء العملية الجارية

The Interlocked Type

يزودنا النوع Interlocked بطريقة للقيام بعمليات دقيقة لزيادة أو إنقاص قيمة متغير مشترك وهذه الفئة تعرض فقط طرائق ساكنة (لا نحتسب هنا ما تمت وراثته من Object) انظر إلى الكود التالي

```
' Increment and Decrement methods work with 32-bit and 64-bit integers.
Dim lockCounter As Integer
…
' Increment the counter and execute some code if its previous value was zero.
If Interlocked.Increment(lockCounter) = 1 Then
   …
End If
' Decrement the shared counter.
Interlocked.Decrement(lockCounter)
```

والطريقة ADD جديدة في الفريمورك 2 وهي تمكنك من زيادة أعداد حقيقية Integer من عيار 32 أو 64 بت بقيمة محددة

```
If Interlocked.Add(lockCounter, 2) <= 10 Then…
```

وتوفر الفئة Interlocked طريقتان ساكنتان أخريان الطريقة Exchange التي تمكنك من تحديد قيمة من اختيارك إلى متغيرات من النوع Integer أو Long أو Single أو Double أو IntPtr أو Object وتعيد القيمة السابقة وبما أن لها نسخة محملة زائدا تأخذ محددا من النوع Object لهذا يمكنك أن تجعلها تعمل لأي نوع مرجعي كالنوع String كما في المثال

```
Dim s1 As String = "123"
Dim s2 As String = Interlocked.Exchange(s1, "abc")
Console.WriteLine("s1={0}, s2={1}", s1, s2)
```

والطريقة CompareExchange تعمل بأسلوب مشابهة ولكنها تقوم بالتبديل فقط إذا كان موقع الذاكرة مساوي لقيمة محددة يتم تمريرها لها

The ManualResetEvent, AutoResetEvent, and EventWaitHandle Types

هذه الفئات الثلاثة تعمل بشكل متشابه ManualResetEvent و AutoResetEvent و EventWaitHandle والفئة الأخيرة هي الفئة الأب للفئتان الأولان وقد تمت إضافتها في الفريمورك 2 على الرغم من أن ManualResetEvent و AutoResetEvent لم يتم إهمالهما بعد وأثناء العمل يمكنك استبدالهما بالفئة الجديدة EventWaitHandle التي تعطيك مزيدا من المرونة عند التعامل. والنوعان ManualResetEvent و AutoResetEvent مفيدان بشكل خاص عندما تريد إيقاف مسار أو أكثر بشكل مؤقت حتى يخبرنا مسار آخر بأنه لا مانع من المتابعة وتستخدمهما لإيقاظ مسار مثل إجراء معالجة الحدث في مسار متوقف ولكن لا تنخدع بوجود Event في أسمائهما فلا يمكنك استخدام إجراءات معالجة الحدث التقليدية مع هذه الأغراض. وكائن من أحد هذين النوعين يمكن أن يكون في حالة إشارة أو عدم إشارة Signale/UnSignaled وهذه القيمة لا تملك أي معنى خاص بحيث يمكنك اعتبارها كحالة تشغيل/إيقاف حيث ستمرر الحالة الابتدائية للباني وأي مسار يستطيع الوصول لذلك الغرض يمكنه ضبط تلك الحالة إلى Signaled باستخدام الطريقة Set أو يستخدم الطريقة Reset لإعادة الحالة إلى UnSignaled ويمكن للمسارات الأخرى استخدام الطريقة WaitOne للانتظار حتى تصبح في حالة إشارة Signaled أو حتى انتهاء فترة الانتظار

```
' Create an auto reset event object in nonsignaled state.
Dim are As New AutoResetEvent(False)
' Create a manual reset event object in signaled state.
Dim mre As New ManualResetEvent(True)
```

والاختلاف الوحيد بين الغرضان ManualResetEvent و AutoResetEvent هو أن الأخير يعيد ضبط نفسه آليا (يصبح في حالة عدم إشارة Unsignaled) وذلك مباشرة بعد أن يتم صد المسار عندما تبدأ الطريقة WaitOne ويوقظ الغرض AutoResetEvent فقط واحد من المسارات المنتظرة عندما يصبح في حالة إشارة بينما الغرض ManualResetEvent يوقظ جميع المسارات المنتظرة ويجب أن يتم إعادة ضبطه يدويا إلى حالة عدم إشارة كما هو ظاهر من اسمه وكما ذكر سابقا يمكنك استبدال الغرضين ManualResetEvent و AutoResetEvent بالغرض EventWaitHandle كما يظهر بالكود التالي

```
' These statements are equivalent to the previous code example.
Dim are As New EventWaitHandle(False, EventResetMode.AutoReset)
Dim mre As New EventWaitHandle(True, EventResetMode.ManualReset)
```

وتكون أغراض الـ Event مفيدة خاصة في حالات المنتج والمستهلك فربما يكون لديك إجراء وحيد في مسار يقوم بتقييم بعض البيانات أو بالقراءة من القرص أو منفذ تسلسلي أو غيرها ويستدعي الطريقة Set على غرض متزامن فيتم إعادة تشغيل مسار أو أكثر لمعالجة تلك البيانات ويجب عليك استخدام الغرض AutoResetEvent أو الغرض EventWaitHandle مع الخيار AutoReset إذا كان هناك مسار مستهلك وحيد سيقوم بمعالجة تلك البيانات كما يجب عليك استخدام الغرض ManualResetEvent أو الغرض EventWaitHandle مع الخيار ManualReset إذا كان يجب معالجة البيانات باستخدام جميع المسارات المستهلكة.

ويبين المثال التالي كيف يمكن أن يكون لديك عدة مسارات منتجة تقوم بعملية البحث عن ملف في عدة مجلدات مختلفة في نفس الوقت ولكن يوجد مسار مستهلك وحيد يقوم بجمع النتائج من تلك المسارات ويستخدم المثال الغرض AutoResetEvent لإيقاظ المسار المستهلك عندما يتم إضافة اسم ملف جديد للقائمة (List(Of String ويستخدم أيضا الفئة Interlocked لإدارة عدد المسارات العاملة حتى يعلم المسار الرئيسي أنه لم تعد توجد أي بيانات أخرى لاستهلاكها

```
' The shared AutoResetEvent object
Public are As New AutoResetEvent(False)
' The list where matching filenames should be added
Public fileList As New List(Of String)()
```

```
' The number of running threads
Public searchingThreads As Integer
' An object used for locking purposes
Public lockObj As New Object()

Sub TestAutoResetEvent()
   ' Search *.zip files in all the subdirectories of C.
   For Each dirname As String In Directory.GetDirectories("C:\")
      Interlocked.Increment(searchingThreads)
      ' Create a new wrapper class, pointing to a subdirectory.
      Dim sf As New FileFinder()
      sf.StartPath = dirname
      sf.SearchPattern = "*.zip"
      ' Create and run a new thread for that subdirectory only.
      Dim t As New Thread(AddressOf sf.StartSearch)
      t.Start()
   Next

   ' Remember how many results we have so far.

   Dim resCount As Integer = 0
   Do While searchingThreads > 0
      ' Wait until there are new results.
      are.WaitOne()

      SyncLock lockObj
         ' Display all new results.
         For i As Integer = resCount To fileList.Count - 1
             Console.WriteLine(fileList(i))
         Next
         ' Remember that you've displayed these filenames.
         resCount = fileList.Count
      End SyncLock
   Loop
   Console.WriteLine("")
   Console.WriteLine("Found {0} files", resCount)
End Sub
```

وكل مسار إجرائي يعمل ضمن غرض FileFinder مختلف الذي يجب أن يكون قادرا على الوصول إلى متغيرات عامة محددة في الكود السابق

```
Class FileFinder
   Public StartPath As String       ' The starting search path
   Public SearchPattern As String   ' The search pattern

   Sub StartSearch()
      Search(Me.StartPath)
      ' Decrease the number of running threads before exiting.
      Interlocked.Decrement(searchingThreads)
      ' Let the consumer know it should check the thread counter.
      are.Set()
   End Sub

   ' This recursive procedure does the actual job.
   Sub Search(ByVal path As String)
      ' Get all the files that match the search pattern.
      Dim files() As String = Directory.GetFiles(path, SearchPattern)
      ' If there is at least one file, let the main thread know about it.
      If files IsNot Nothing AndAlso files.Length > 0 Then
         ' Ensure found files are added as an atomic operation.
         SyncLock lockObj
            ' Add all found files.
```

```
            fileList.AddRange(files)
            ' Let the consumer thread know about the new filenames.
            are.Set()
         End SyncLock
      End If

      ' Repeat the search on all subdirectories.
      For Each dirname As String In Directory.GetDirectories(path)
         Search(dirname)
      Next
   End Sub
End Class
```

والنسخة 2 من الفريموورك وفيجول بايزيك 2005 تستخدم EventWaitHandle بدلا عن AutoResetEvent أو ManualResetEvent مما يعطيك ميزة هامة وهي إمكانية إنشاء غرض مسمى على مستوى النظام يمكن مشاركته مع العمليات الأخرى والصيغة العامة لباني EventWaitHandle مشابهة لتلك الخاصة بالفئة Mutex

```
Create a system-wide auto reset event that is initially in the signaled state.
Dim ownership As Boolean
Dim ewh As New EventWaitHandle(True, EventResetMode.AutoReset, "eventname",
ownership)
If ownership Then
   ' The event object was created by the current thread.
   …
End If
```

كما يمكنك استخدام الطريقة OpenExisting لفتح غرض حدث موجود

```
' This statement throws a WaitHandleCannotBeOpenedException if the specified
' event doesn't exist, or an UnauthorizedAccessException if the current
' user doesn't have the required permissions.
ewh = EventWaitHandle.OpenExisting("eventname", _
               EventWaitHandleRights.FullControl)
```

والميزة الأخرى الهامة في أغراض الأحداث event objects في الفريموورك 2 هي دعم ACLs باستخدام الطرائق SetAccessControl و GetAccessControl والتي تأخذ وتعيد كائن من النوع EventWaitHandleSecurity حيث يمكنك استخدامه بنفس طريقة استخدام مثيلاتها في الغرض Mutex وأغراض الدوت نيت الأخرى التي تدعم ACLs

كيفية تنفيذ عملية في مسار آخر وإظهار النتيجة في التحكمات على النموذج

سألني أحد الإخوة عن مشكلة واجهته عند تنفيذ عملية معينة على مسار آخر ومحاولته إظهار النتيجة على النموذج فإذا افترضنا أنه لدينا إجراء بسيطا ينفذ عملية ما ونفذناه على مسار آخر Thread غير المسار الرئيسي الذي تنفذ عليه عمليات البرنامج وأن ذلك الإجراء يحتوي على كود يقوم بضبط قيمة الخاصية Text لصندوق نصوص على النموذج فعند تنفيذ الكود ستحصل على رسالة خطأ

Cross-thread operation not valid: Control 'TextBox1' accessed from a thread other than the thread it was created on.

وإذا أردت توليد رسالة الخطأ السابق بنفسك أنشئ مشروعا جديدا ضع عليه صندوق نصوص وزر واجعل كود النموذج مطابقا لما يلي ثم قم بتشغيل البرنامج وستحصل على رسالة الخطأ السابقة

```
Imports System.Threading

Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click

        Dim th As New Thread(AddressOf DoLongOperation)
        th.Start()

    End Sub

    Private Sub DoLongOperation()
        Me.TextBox1.Text = "Something"
    End Sub

End Class
```

الحل الذي أقوم باستخدامه عادة لحل هكذا مشكلة هو إنشاء فئة Class تقوم بتنفيذ العملية على المسار الثاني وتعيد النتيجة للنموذج من خلال إطلاق حدث يعيد القيم الناتجة عن عملية المعالجة للنموذج فربما لا تكون هذه هي الطريقة الأفضل في جميع الحالات ولكنها طريقتي على كل حال وسأقوم بشرحها ثم يمكننا النقاش وتجربة أية حلول أخرى لتجاوز هذه المشكلة وسأطرحها عبر تنفيذ عداد بسيط للوقت فربما ستستخدم أنت هذه الطريقة للبحث عن ملفات أو تنفيذ عمليات معالجة معقدة تستغرق وقتا طويلا ولكنني هنا اخترت مثالا يعيد قيمة وحيدة بحيث يكون بسيطا قدر الإمكان

الآن سأقوم بإضافة فئة Class جديد للمشروع يتم عبره تنفيذ العملية الطويلة التي نريد تنفيذها على مسار آخر وسأقوم بتسميتها MyStopWatch في الوقت الحالي وبما أننا سنتعامل مع المسارات سنحتاج للاستيراد التالي قبل تعريف الفئة

```
Imports System.Threading
```

سأقوم بتعريف فئة فرعية داخل الفئة MyStopWatch باسم ReturnValueEventArgs سأستخدمها لاحقا لإطلاق الحدث الذي سيعيد النتيجة إلى النموذج وهذه يجب أن تكون موروثة من الفئة EventArgs بما أنها فئة خاصة بإعادة قيم الحدث الذي سيتم إطلاقه وسأعرف فيها خاصية وحيدة ReturnVlaue ستكون للقراءة فقط بما أننا لن نحتاج لضبط قيمتها إلا من خلال باني الفئة تعيد القيمة وباني للفئة يمرر له قيمة نصية وحيدة تمثل القيمة المعادة وبهذا يكون كود الفئة ReturnValueEventArgs كما يلي

```
Public Class ReturnValueEventArgs
    Inherits EventArgs

    Private _ReturnValue As String

    Public ReadOnly Property ReturnVlaue() As String
        Get
            Return _ReturnValue
        End Get
    End Property
```

```
    Public Sub New(ByVal RetVal As String)
        _ReturnValue = RetVal
    End Sub

End Class
```

ضمن كود الفئة MyStopWtach وبعد نهاية تعريف الفئة ReturnValueEventArgs نقوم بتعريف الحدث الذي سنقوم بإطلاقه ليعيد القيمة إلى النموذج ومن أجل الالتزام بتنسيق الأحداث كما نرى في التحكمات والفئات قمنا بتعريف الفئة ReturnValueEventArgs وبهذا يكون تعريف الحدث في قسم تعريف المتغيرات العامة في الفئة MyStopWatch كما يلي

```
Public Event ReturnValue(ByVal sender As Object, ByVal e As ReturnValueEventArgs)
```

عرف متغيرا عاما على مستوى الفئة MyStopWtach باسم `_MyTimer` وهو من النوع Stopwatch كما يلي

```
Private _MyTimer As Stopwatch
```

حيث سنستخدمه كعداد للوقت من أجل الحصول على قيمة ليتم إعادتها ضمن إجراء المعالجة الذي سيتم تنفيذه على المسار الآخر بحيث سيكون كود إجراء المعالجة الذي سينفذ على المسار الثاني كما يلي

```
Private Sub DoProcessing()
    Do
        Dim Ret = _MyTimer.Elapsed.Hours & ":" & _
                  _MyTimer.Elapsed.Minutes & ":" & _
                  _MyTimer.Elapsed.Seconds & ":" & _
                  _MyTimer.Elapsed.Milliseconds

        RaiseEvent ReturnValue(Me, New ReturnValueEventArgs(Ret))
    Loop Until _MyTimer.IsRunning = False
End Sub
```

حيث وضعنا قيمة العداد في متغير نصي Ret ثم استخدمنا الدالة RaiseEvent لإطلاق الحدث ReturnValue حيث القيمة Me التي تشير للفئة الحالية كبارمتر أول للحدث ReturnValue يكون المحدد الثاني للحدث عبارة عن كيان Instance من الفئة ReturnValueEventArgs التي نمرر لبانيها المتغير Ret الذي يشكل القيمة المعادة من الخاصية ReturnVlaue العائدة للفئة ReturnValueEventArgs عندما سنستقبلها من النموذج

وسيكون لدينا إجراء لبدء تنفيذ المؤقت على المسار الثاني باسم StartTimer بحيث يكون كوده على الشكل

```
Public Sub StartTimer()
    _MyTimer.Reset()
    _MyTimer.Start()

    Dim th As New Thread(AddressOf DoProcessing)
    th.Start()
End Sub
```

حيث قمنا بتصفير العداد وبدئه ثم عرفنا مسارا جديدا th يقوم بتنفيذ الإجراء DoProcessing ومن أجل إيقاف العداد سنحتاج لإجراء StopTimer يكون كوده على الشكل

```
Public Sub StopTimer()
    _MyTimer.Stop()
End Sub
```

وبهذا تكون قد اكتملت فئتنا التي ستقوم بعملية المعالجة على مسار ثاني وتعيد قيمة نصية سنقوم بعرضها في صندوق نصوص لاحقا ويكون بذلك الكود الكامل لهذه الفئة

```
Imports System.Threading

Public Class MyStopWtach

    Public Class ReturnValueEventArgs
        Inherits EventArgs

        Private _ReturnValue As String

        Public ReadOnly Property ReturnVlaue() As String
            Get
                Return _ReturnValue
            End Get
        End Property

        Public Sub New(ByVal RetVal As String)
            _ReturnValue = RetVal
        End Sub

    End Class

    Public Event ReturnValue(ByVal sender As Object, _
                  ByVal e As ReturnValueEventArgs)

    Private _MyTimer As New Stopwatch

    Public Sub StartTimer()
        _MyTimer.Reset()
        _MyTimer.Start()

        Dim th As New Thread(AddressOf DoProcessing)
        th.Start()
    End Sub

    Private Sub DoProcessing()
        Do
            Dim Ret = _MyTimer.Elapsed.Hours & ":" & _
                      _MyTimer.Elapsed.Minutes & ":" & _
                      _MyTimer.Elapsed.Seconds & ":" & _
                      _MyTimer.Elapsed.Milliseconds

            RaiseEvent ReturnValue(Me, New ReturnValueEventArgs(Ret))
        Loop Until _MyTimer.IsRunning = False
    End Sub

    Public Sub StopTimer()
        _MyTimer.Stop()
    End Sub


End Class
```

نعود للنموذج الخاص بالمشروع الذي نحتاج لوجود صندوق نصوص وزرين عليه للقيام بتجربة الفئة الجديدة حيث سنقوم بتعريف متغير خاص على مستوى النموذج باسم MyTimer من نوع فئتنا MyStopWtach وباستخدام العبارة WithEvents التي ستمكننا من استقبال الأحداث التي ستطلقها فئتنا

```
Private WithEvents MyTimer As New MyStopWtach
```

وسيكون كود الزرين من أجل بدء وإيقاف المؤقت باستخدام فئتنا السابقة كما يلي

```
Private Sub Button1_Click() Handles Button1.Click
    MyTimer.StartTimer()
```

```
End Sub

Private Sub Button2_Click() Handles Button2.Click
    MyTimer.StopTimer()
End Sub
```

الآن أنشئ معالج للحدث ReturnVlaue العائد للمتغير MyTimer واجعله بحيث يكون الكود فيه كالتالي ثم جرب تشغيل البرنامج فستحصل على رسالة مشابهة للرسالة في بداية المقال

```
Private Sub MyTimer_ReturnValue(ByVal sender As Object, _
      ByVal e As MyStopWtach.ReturnValueEventArgs) Handles MyTimer.ReturnValue

    Me.TextBox1.Text = e.ReturnVlaue

End Sub
```

ولمعالجة هذه النقطة والتخلص من رسالة الخطأ سنحتاج لعمل Invoke للإجراء MyTimer_ReturnValue حتى نستطيع استخدام القيم المعادة منه في ضبط قيم خصائص التحكمات على النموذج وفي حالتنا هنا الخاصية Text لصندوق النصوص والعملية ببساطة ستتم كما يلي

في قسم المتغيرات العامة في النموذج سنقوم بتعريف إجراء مفوض Delegate يحمل نفس توقيع الإجراء MyTimer_ReturnValue وبدون جسم للإجراء كما يلي

```
Private Delegate Sub MyTimer_ReturnValueDelegate(ByVal sender As Object, _
                             ByVal e As MyStopWtach.ReturnValueEventArgs)
```

ويكون الكود الذي سينفذ المهمة بصورة صحيحة كما يلي

```
Private Sub MyTimer_ReturnValue(ByVal sender As Object, _
        ByVal e As MyStopWtach.ReturnValueEventArgs) Handles MyTimer.ReturnValue

    If Me.TextBox1.InvokeRequired = True Then
        Dim d As New MyTimer_ReturnValueDelegate(AddressOf MyTimer_ReturnValue)
        Me.Invoke(d, New Object() {sender, e})
    Else
        Me.TextBox1.Text = e.ReturnVlaue
    End If
End Sub
```

حيث فحصنا قيمة الخاصية القابلة للقراءة فقط InvokeRequired لصندوق النصوص فإن كان False نقوم بضبط قيمة الخاصية Text باستخدام القيمة المعادة من الحدث مباشرة بدون أي مشاكل وإن كانت True عندها لن نستطيع ضبط القيمة مباشرة كي لا نحصل على الخطأ الوارد في بداية المقال عندها سنعرف متغير d من نوع الإجراء المفوض MyTimer_ReturnValueDelegate ونمرر له عنوان الإجراء MyTimer_ReturnValue ثم استخدمنا الطريقة Invoke العائدة للنموذج لتنفيذ نسخة آمنة من الحدث MyTimer.ReturnValue تمكننا من ضبط القيم المعادة إلى التحكمات وذلك بتمرير المتغير d كمحدد أول للخاصية Invoke ويكون المحدد الثاني للخاصية Invoke هو مصفوفة من النوع Object يتم تمرير محددات الإجراء MyTimer_ReturnValue كعناصر لها

وفيما يلي الكود الكامل للنموذج

```
Public Class Form1

    Private Delegate Sub MyTimer_ReturnValueDelegate(ByVal sender As Object, _
                                 ByVal e As MyStopWtach.ReturnValueEventArgs)

    Private WithEvents MyTimer As New MyStopWtach

    Private Sub Button1_Click() Handles Button1.Click
        MyTimer.StartTimer()
```

```
    End Sub

    Private Sub Button2_Click() Handles Button2.Click
        MyTimer.StopTimer()
    End Sub

    Private Sub MyTimer_ReturnValue(ByVal sender As Object, _
             ByVal e As MyStopWtach.ReturnValueEventArgs) Handles _
              MyTimer.ReturnValue

        If Me.TextBox1.InvokeRequired = True Then
            Dim d As New MyTimer_ReturnValueDelegate( _
             AddressOf MyTimer_ReturnValue)

            Me.Invoke(d, New Object() {sender, e})
        Else
            Me.TextBox1.Text = e.ReturnVlaue
        End If
    End Sub

End Class
```